HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

# Last Chapter:
# Course Rehash

Fall 2012

Lecturer: Sini Ruohomaa

(Slides copied/summarized from other course material.)

# Definition and Goals of Distributed Systems

- *Collection of independent computers – appears to users as single coherent system*
- Goals:
  - Making resources accessible
  - Openness
  - Scalability
  - Security
  - Fitting the given concrete environment
  - Fulfilling system design requirements
  - Distribution transparency
- Challenges with all of these (see Chapter 1)

# Transparencies (RM-ODP standard, 1998)

| Transparency | Description |
|---|---|
| Access | Hide differences in data representation and how a resource is accessed |
| Location | Hide where a resource is located (∗) |
| Migration | Hide that a resource may move to another location (∗)<br>(the resource does not notice) |
| Relocation | Hide that a resource may be moved to another location (∗)<br>while in use (the others don't notice) |
| Replication | Hide that a resource is replicated |
| Transaction | Hide that multiple competing users perform concurrent actions on the resource |
| Failure | Hide the failure and recovery of a resource |
| Persistence | Hide whether a (software) resource is in memory or on disk |

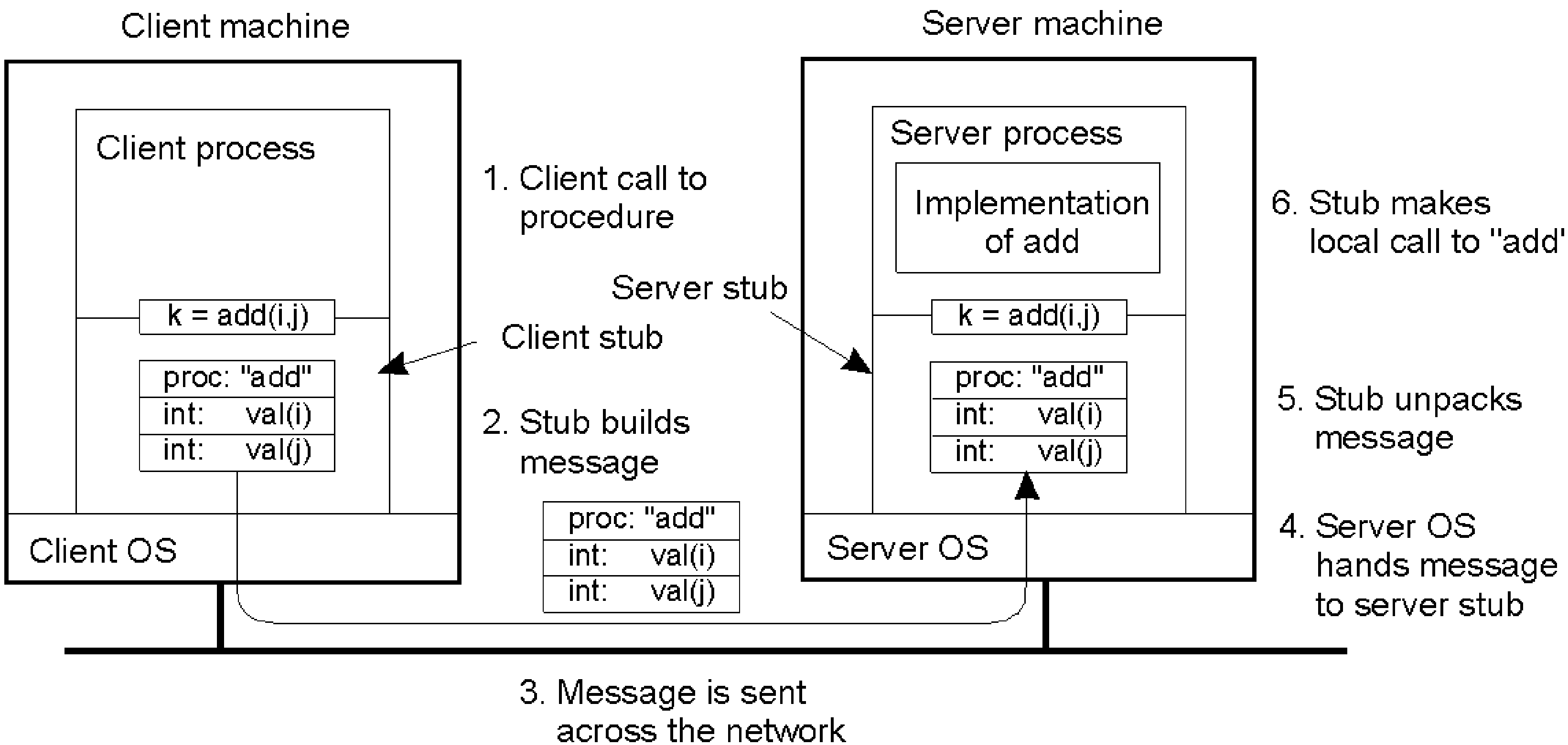
(∗) Note the various meanings of "location": network address (several layers) ; geographical address

# False assumptions everyone makes when developing their first distributed application:

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwith is infinite
- Transport cost is zero
- There is one administrator
- There is inherent, shared knowledge

- By Peter Deutsch

# Remote Procedure Calls (RPC)

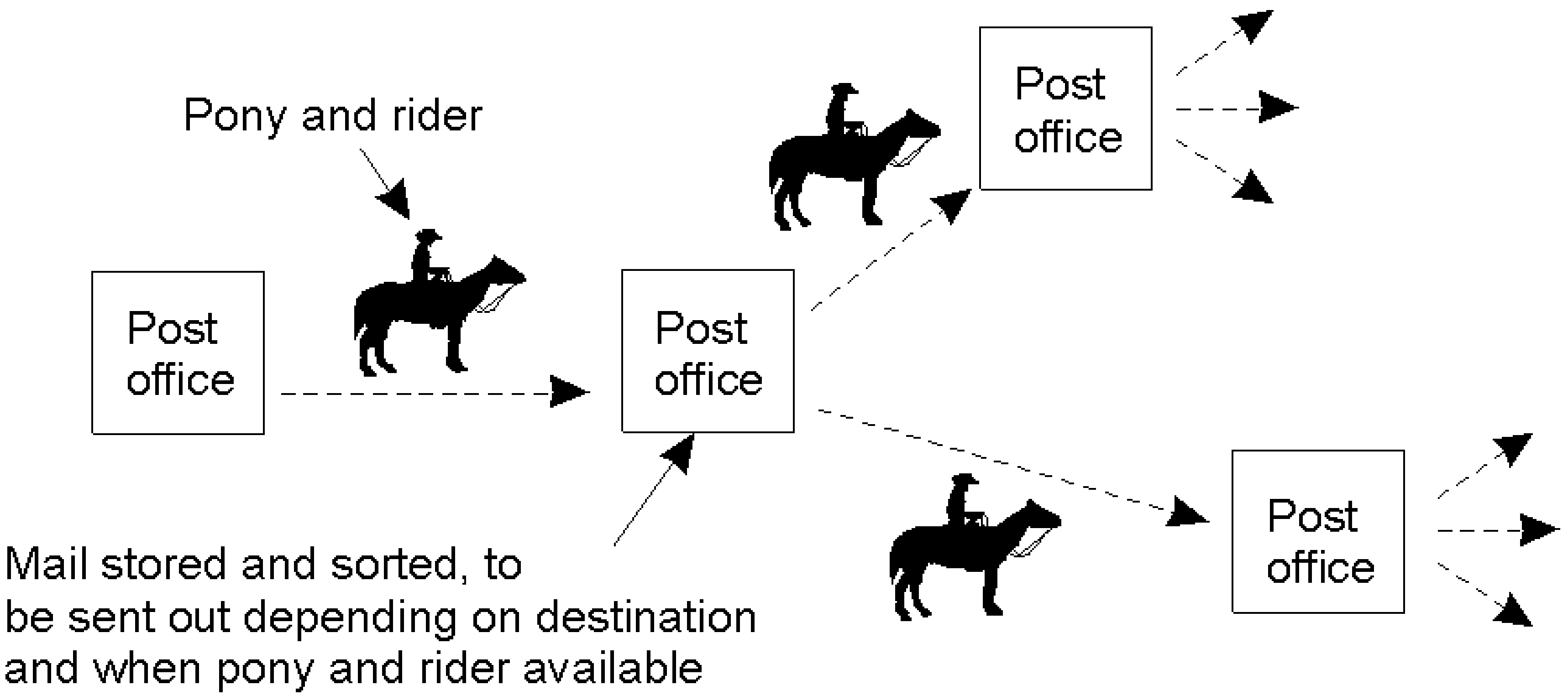



Steps involved in doing remote computation through RPC

# RPC Design Issues

- Delivery guarantees: RPC/RMI failure semantics
  - Maybe (no retransmit)
  - At-least-once (retransmit + re-execute)
  - At-most-once (retransmit + duplicate filtering to not redo)
  - (Un-achievable: exactly-once)
- Handling exceptions
- Transparency (algorithmic vs. behavioral)

# Persistence and Synchronicity in Communication




Persistent communication of letters back in the days of the Pony Express.

# Time and Clocks

| What we need? | How to solve? |
|---|---|
| Real time (17:30:21) | Universal time<br>(- Synchronize clocks!) |
| Interval length (3 ms) | Computer clock |
| Order of events (1.,2.) | Logical clocks<br>(Universal time) |

NOTE: ***Time*** is ***monotonous***

# Synchronization of Clocks: Software-Based Solutions

- Techniques:
  - time stamps of real-time clocks
  - message passing
  - round-trip time (local measurement)
- Cristian’s algorithm – ask centralized clock
- Berkeley algorithm – synchronized within a group
- NTP: Network time protocol (Internet)

# Example Problem: Totally-Ordered Multicasting (3)

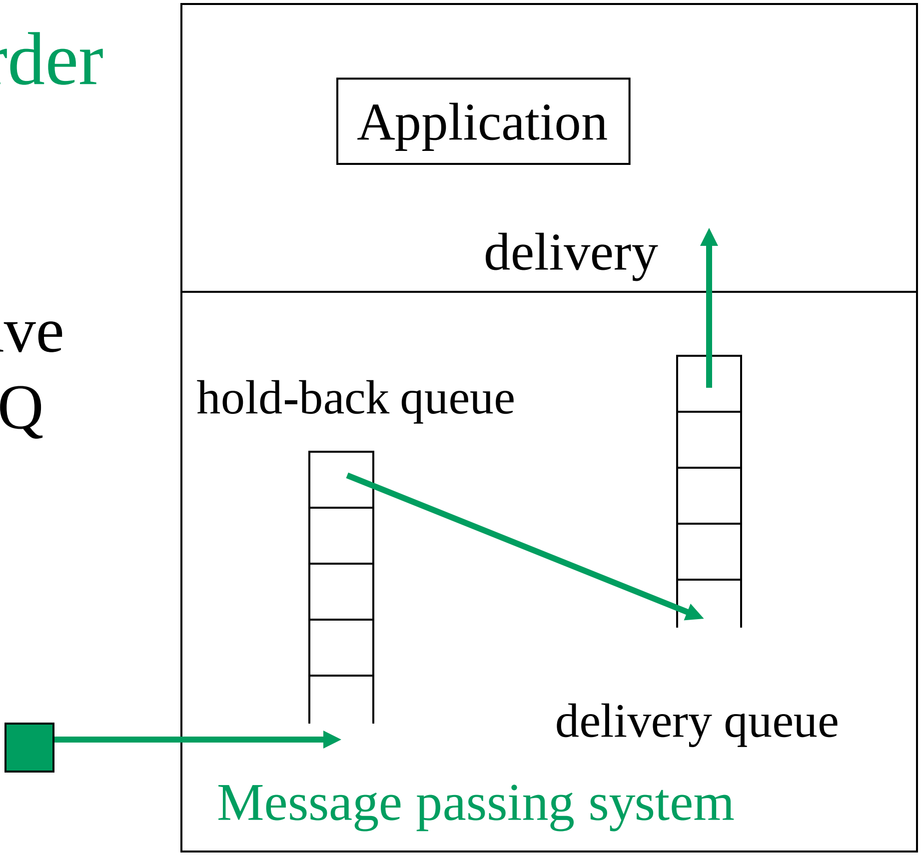
Guaranteed delivery order

- *new* message => HBQ

- when *all predecessors* have arrived: message => DQ

**How to detect this?**

- when *at the head of DQ*: message => application (application: *receive* …)



# Logical Clocks: Vector Timestamps

**Goal**:

timestamps should reflect *causal ordering*

L(e) < L(e') => " e happened before e' "

**=>**

**Vector clock**

each process $P_i$ maintains a vector $V_i$ :

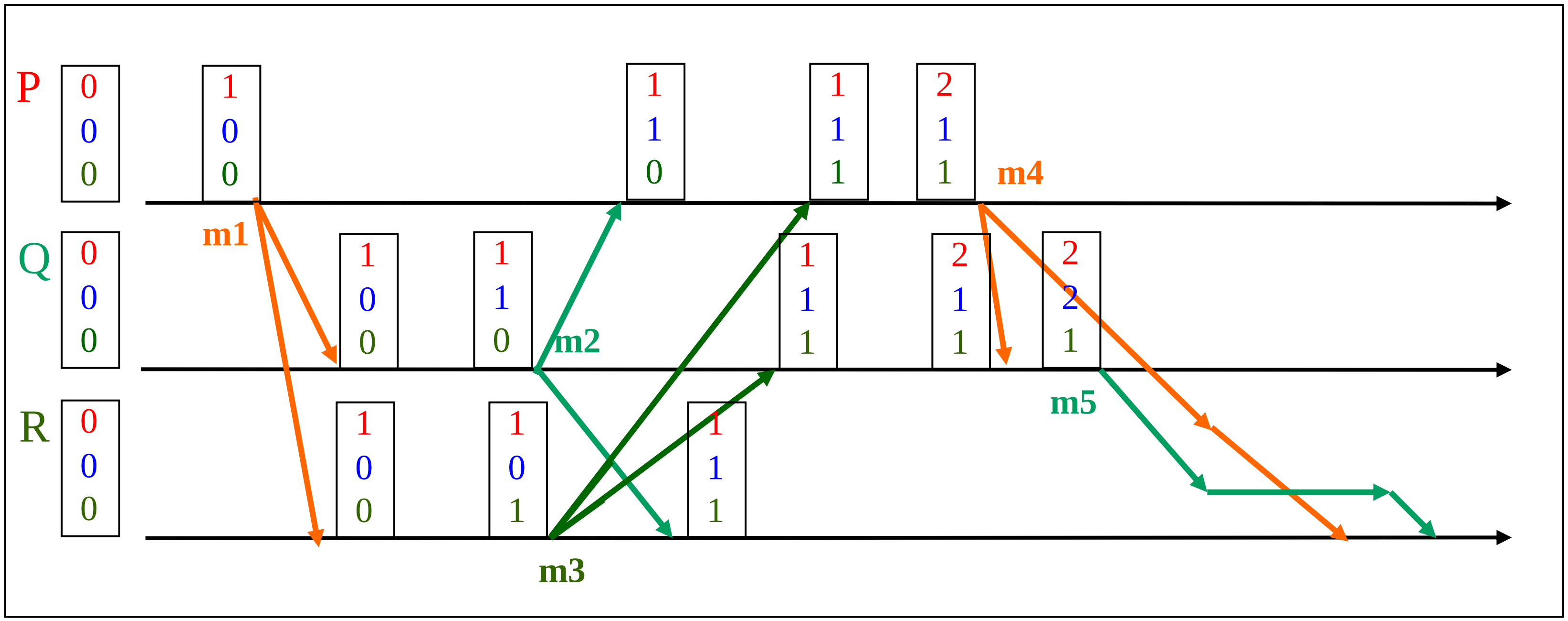
1. $V_i[i]$ is the number of events that have occurred at $P_i$

   *(the current local time at $P_i$ )*

2. if $V_i[j] = k$ then $P_i$ knows about (the first) k events that have occurred at $P_j$

   *(the local time at $P_j$ was k, as $P_j$ sent the last message that $P_i$ has received from it)*

# Causal Ordering of Multicasts (1)




**Event**:
message sent

**Timestamp [i,j,k] :**

i messages sent from P

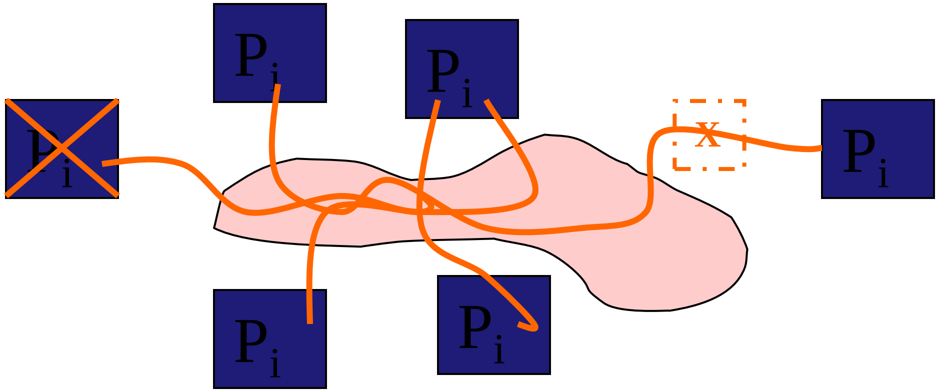
j messages sent form Q

k messages sent from R

R: m1 [100] m4 [211]
m2 [110] m5 [221]
m3 [101]

m4 [211] vs. 111

# Coordination and Agreement




- Reserving resources *(distributed mutual exclusion) :*
  - Centralized, Ricart-Agrawala, Token ring
- Elections (electing coordinator, initiator): Bully algorithm, Ring algorithm
- Multicasting: a sensibly ordered reliable multicast would be nice (see ch. 5)
- Distributed transactions: snapshots/checkpointing, two-phase commit

# Reasons for Data Replication

- **Dependability requirements**
  - availability
    - at least some server somewhere
    - wireless connections => a local cache
  - reliability (correctness of data)
    - fault tolerance against data corruption
    - fault tolerance against faulty operations
- **Performance**
  - response time, throughput
  - scalability
    - increasing workload
    - geographic expansion
  - mobile workstations => a local cache
- **Price to be paid: consistency maintenance**
  - performance vs. required level of consistency (need not care ⇔ updates immediately visible)

# Consistency: Data-Centric Consistency Models (1)

| Consistency | Description |
|---|---|
| Strict | Absolute time ordering of all shared accesses matters. |
| Linearizability | All processes see all shared accesses in the same order. Accesses are furthermore ordered according to a (nonunique) global timestamp |
| Sequential | All processes see all shared accesses in the same order. Accesses are not ordered in time |
| Causal | All processes see causally-related shared accesses in the same order. |
| FIFO | All processes see writes from each other in the order they were performed. Writes from different processes may not always be seen in the same order by other processes. |

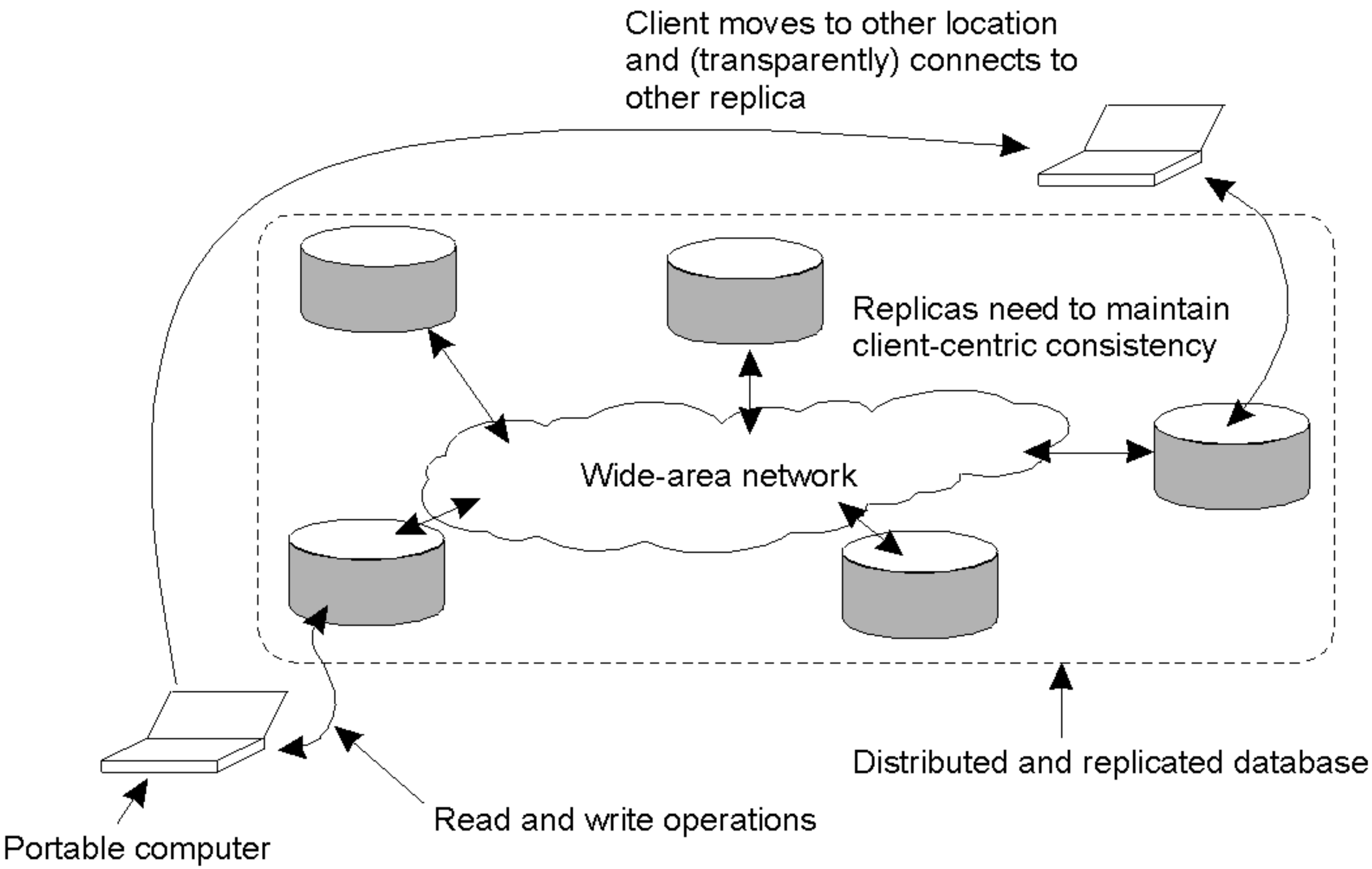
*Consistency models at the level of read and write operations. Next: grouping operations.*

## Summary of Consistency Models (2)

| Consistency | Description |
| --- | --- |
| Entry | Shared data associated with a synchronization variable are made consistent when a critical section is entered. |
| Release | All shared data are made consistent after the exit out of the critical section (and up-to-dateness checked upon entry) |
| Weak | Shared data can be counted on to be consistent only after an explicit synchronization is done |

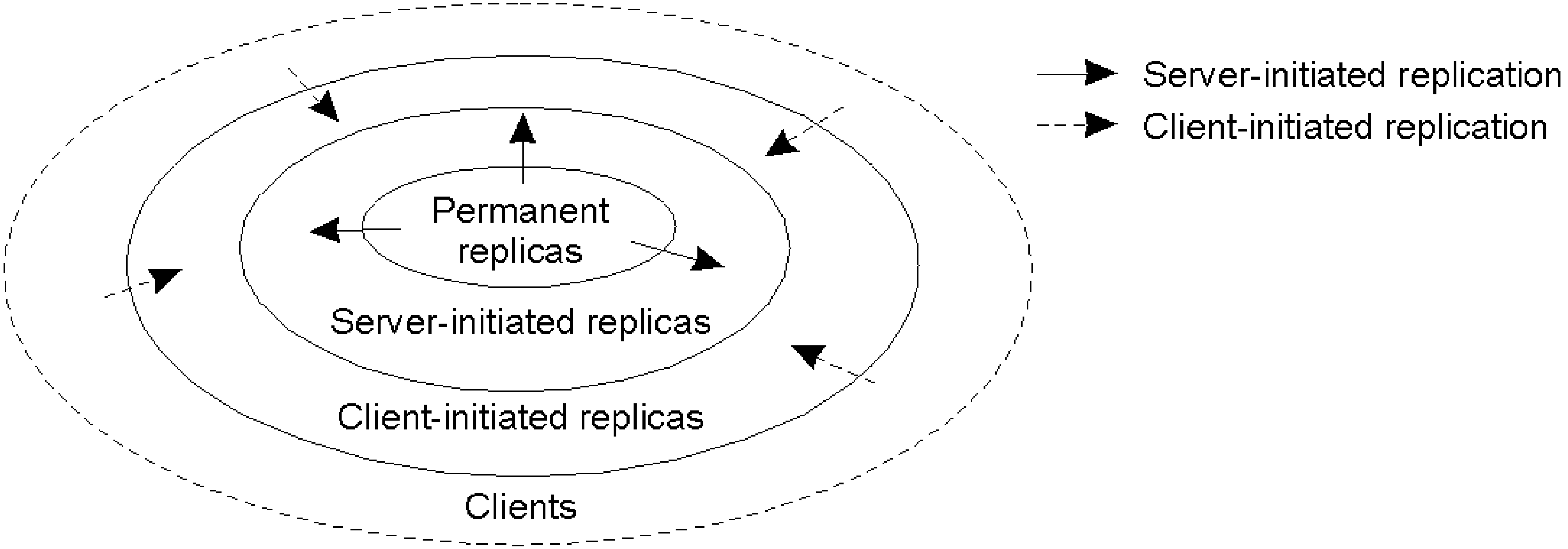
*Models built around grouping operations and synchronization.*

# Client-Centric Consistency




Monotonic reads, Monotonic Writes, Read your Writes, Writes Follow Reads

# Replica Placement (1)




The logical organization of different kinds of copies of a data store into three concentric rings.

# Update Propagation to Replicas

- Update route: client writes to copy, who writes to {other copies}
- Whose responsibility – “push” or “pull”?
- Issues:
  - Consistency of copies
  - Cost: traffic, maintenance of state data
- What information is propagated?
  - Notification of an update (**invalidation** protocols)
  - Transfer of data itself or diff (useful if high reads-to-writes ratio)
  - Propagate the update operation: e.g. order_flight(x,y)

    (**active replication**)

# Epidemic Protocols

- Example: Epidemic protocols (ch. 4.5)
  - A node with an update: **infective**
  - A node not yet updated: **susceptible**
  - A node not willing / able to spread the update: **removed**
  - **Propagation protocol example**: **anti-entropy**
    - Node **P** picks randomly another node **Q**, and…
    - Three information exchange alternatives:

      **P** pushes to **Q** or **P** pulls from **Q** or **P**⇔**Q** push-pull
    - Push good early, pull when many infected, push-pull best
  - **Variant of this**: **gossiping**

# Consistency Protocols

- Consistency protocol: implementation of a consistency model
- The most widely applied models
  - Sequential consistency
  - Weak consistency with synchronization variables
  - Atomic transactions (cf. ACID properties)
- The main approaches
  - **Primary-based protocols** (remote write, local write)
  - Replicated-write protocols (**active replication**, **quorum** based)
  - (Cache-coherence protocols)

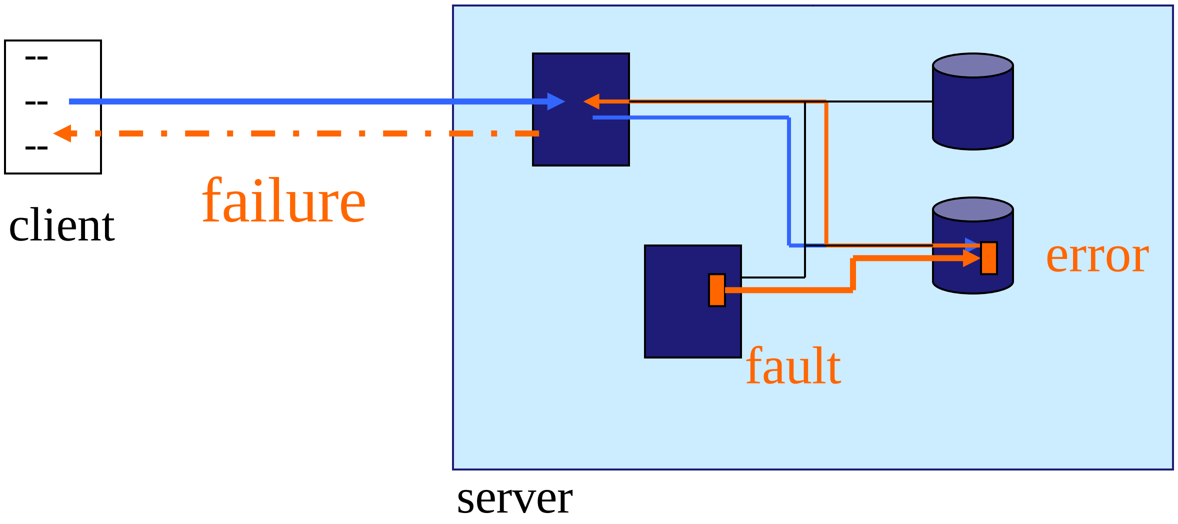
# Fault Tolerance Basic Concepts

Dependability includes

- Availability – system can be used immediately
- Reliability – runs continuously without failure
- Safety – failures do not lead to disaster
- Maintainability – recovery from failure is easy

Note: security is a separate issue from these.

# Basic concepts: Fault, error, failure




- Fault: e.g. bad design/bug/physical limitation (causes error)
- Error: system state is incorrect (may lead to failure)
- Failure: cannot meet promises (e.g. full delivery of service)

# Failure models: Different types of failures

| Type of failure | Description |
|---|---|
| Crash failure | A server halts, but is working correctly until it halts |
| Omission failure | A server fails to respond to incoming requests |
| *Receive omission* | A server fails to receive incoming messages |
| *Send omission* | A server fails to send messages |
| Timing failure | A server's response lies outside the specified time interval |
| Response failure | The server's response is incorrect |
| *Value failure* | The value of the response is wrong |
| *State transition failure* | The server deviates from the correct flow of control |
| Arbitrary failure (= Byzantine failure) | A server may produce arbitrary responses at arbitrary times |

Crash: fail-stop, fail-safe (*detectable*), fail-silent (*seems to have crashed*)

# Failure models: Timing failures

| *Class of Failure* | *Affects* | *Description* |
|---|---|---|
| Clock | Process | Process's local clock exceeds the bounds on its rate of drift from real time. |
| Performance | Process | Process exceeds the bounds on the interval between two steps. |
| Performance | Channel | A message's transmission takes longer than the stated bound. |

# Summary of Fault Tolerance Methods

- Failure detection, failure masking (quiet recovery)
  - Forward and backward recovery
  - Distributed snapshots, coordinated checkpointing, message logging and “replaying events”
- Process resilience
  - Voting, Byzantine generals
  - Primary with hot and cold standby
- Reliable communication, e.g. reliable multicast
  - Handling group changes (how to find out?)
  - Virtual synchrony

# This Course In a Nutshell

- What is distribution and a distributed system?
  - Reasons? Goals? Challenges?
- Distributed decision-making and communication:
  - Working together: clocks, mutual exclusion, elections, transactions
- Replication: Why? How to handle updates and consistency? What kind of consistency needed?
- Fault tolerance: What to do when things go wrong? How to prepare for it?

- Don’t forget real-world applicability!
  - Where to simplify? What to assume?